Приложения на сортиращите алгоритми

Тодор Дуков

Обща информация за някои алгоритми за сортиране

Понякога се оказва много удобно да сортираме входните данни, защото това ни носи полезна информация. Затова е хубаво да се знаят различните алгоритми за сортиране и в какво те са добри. Нека ги сравним по тяхната сложност, като:

- $T_{avg}(n)$ е сложността по време в средния случай
- $M_{avg}(n)$ е сложността по памет в средния случай
- $T_{worst}(n)$ е сложността по време в най-лошия случай
- $M_{worst}(n)$ е сложността по памет в най-лошия случай

име	$T_{avg}(n)$	$M_{avg}(n)$	$T_{worst}(n)$	$M_{worst}(n)$
пирамидално сортиране	$\Theta(n\log(n))$	$\Theta(1)$	$\Theta(n\log(n))$	$\Theta(1)$
сортиране чрез сливане	$\Theta(n\log(n))$	$\Theta(n)$	$\Theta(n\log(n))$	$\Theta(n)$
бързо сортиране	$\Theta(n\log(n))$	$\Theta(\log(n))$	$\Theta(n^2)$	$\Theta(n)$

Въпреки че алгоритъмът за бързо сортиране е по-бавен в най-лошия случай от пирамидалното сортиране и сортирането чрез сливане, практически се оказва, че се справя по-добре. Разбира се, другите сортировки също имат предимства. Пирамидалното сортиране заема малко памет, което е много полезно във вградени системи. Сортирането чрез сливане се оказва по-бързо, когато данните са много големи и се съхраняват във външни устройства (да кажем, на твърд диск). Освен тези алгоритми има и други хубави алгоритми като сортиране чрез броене, но за съжаление ние няма да ги разглеждаме тук.

Два алгоритъма

Ще започнем с два често срещани алгоритъма, които се възползват от това, че данните идват сортирани:

- двоично търсене
- алгоритъма за задачата 2-sum

Нека започнем с алгоритъма за двоично търсене:

```
int binary_search(int *arr, int n, int val)
{
    int left = 0, right = n - 1;

    while (left <= right)
    {
        int mid = left + (right - left) / 2;

        if (arr[mid] == val)
            return mid;
        else if (arr[mid] < val)
            left = mid + 1;
        else
            right = mid - 1;
    }

    return -1;
}
```

При подаден сортиран масив от числа $\texttt{arr}$ с размер $\texttt{n}$ и стойност $\texttt{val}$, функцията $\texttt{binary\_search}(\texttt{arr}, \texttt{n}, \texttt{val})$ ще върне индекс на $\texttt{arr}$, в който се намира $\texttt{val}$, ако има такъв, иначе ще върне -1:

Инвариант. *При всяко достигане на проверката за край на цикъла на ред 5 имаме, че стойността* $\texttt{val}$ *не се намира измежду двата масива* $\texttt{arr}[0 \ldots \texttt{left} - 1]$ *и* $\texttt{arr}[\texttt{right} + 1 \ldots \texttt{n} - 1]$.

База. При първото достигане имаме, че $\texttt{left} = 0$ и $\texttt{right} = \texttt{n} - 1$. Наистина стойността $\texttt{val}$ не се намира измежду двата масива $\texttt{arr}[0 \ldots 0 - 1]$ и $\texttt{arr}[\texttt{n} - 1 + 1 \ldots \texttt{n} - 1]$.
Поддръжка. Нека твърдението е изпълнено за някое непоследно достигане на проверката за край на цикъла. Тогава $\texttt{val}$ не се намира измежду $\texttt{arr}[0 \ldots \texttt{left} - 1]$ и $\texttt{arr}[\texttt{right} + 1 \ldots \texttt{n} - 1]$, и понеже достигането е непоследно, $\texttt{left} \leq \texttt{right}$. Тогава $\texttt{mid} = \left\lfloor \frac{\texttt{left}+\texttt{right}}{2} \right\rfloor$, откъдето $\texttt{left} \leq \texttt{mid} \leq \texttt{right}$. Трябва да разгледаме следните три случая:

1 сл. $\texttt{arr}[\texttt{mid}] = \texttt{val}$ – това няма как да е изпълнено понеже достигането е нефинално

2 сл. $\texttt{arr}[\texttt{mid}] < \texttt{val}$ – понеже $\texttt{arr}$ е сортиран, няма как $\texttt{val}$ да се намира измежду $\texttt{arr}[0 \ldots \texttt{mid}]$, откъдето $\texttt{val}$ не се намира измежду $\texttt{arr}[0 \ldots \underbrace{\texttt{mid} + 1}_{\texttt{left}_{\texttt{new}}} - 1]$ и $\texttt{arr}[\texttt{right} + 1 \ldots \texttt{n} - 1]$.

3 сл. $\texttt{arr}[\texttt{mid}] > \texttt{val}$ – напълно дуален на 2 сл.

Терминация. От цикъла винаги ще излезем, защото или ще открием $\texttt{val}$, или $\texttt{right} - \texttt{left}$ ще намалява, докато не стане отрицателна. Излизането става по два начина:

- не е изпълнено условието на ред 5 т.е. $\texttt{left} > \texttt{right}$ – тогава $\texttt{left} - 1 \geq \texttt{right}$ и понеже $\texttt{val}$ не се намира измежду $\texttt{arr}[0 \ldots \texttt{left} - 1]$ и $\texttt{arr}[\texttt{right} + 1 \ldots \texttt{n} - 1]$, $\texttt{val}$ не се намира във $\texttt{arr}[0 \ldots \texttt{n} - 1]$. Накрая алгоритъмът ще върне -1, което наистина е желания резултат.

- изпълнено е условието на ред 9 т.е. $\texttt{arr}[\texttt{mid}] = \texttt{val}$ – тогава алгоритъмът коректно връща $\texttt{mid}$.

Нека сега разгледаме алгоритъма за задачата 2-sum:

```
1  bool two_sum(int *arr, int n, int target)
2  {
3      int left = 0, right = n - 1;
4
5      while (left < right)
6      {
7          if (arr[left] + arr[right] == target)
8              return true;
9          else if (arr[left] + arr[right] < target)
10             ++left;
11         else
12             --right;
13     }
14
15     return false;
16 }
```

Ще покажем, че при подаден сортиран масив $\texttt{arr}$ с размер $\texttt{n}$ и число $\texttt{target}$, функцията $\texttt{two\_sum}(\texttt{arr}, \texttt{n}, \texttt{target})$ разпознава дали има $0 \leq \texttt{i} < \texttt{j} \leq \texttt{n} - 1$, за които:

$$\texttt{arr}[\texttt{i}] + \texttt{arr}[\texttt{j}] = \texttt{target} \quad // \text{ всяка такава двойка } (\texttt{arr}[\texttt{i}], \texttt{arr}[\texttt{j}]) \text{ ще наричаме } \textit{диада}$$

Инвариант. *При всяко достигане на проверката за край на цикъла на ред 5, всички диади са в* $\texttt{arr}[\texttt{left} \ldots \texttt{right}]$.

База. При първото достигане имаме, че $\texttt{left} = 0$ и $\texttt{right} = \texttt{n} - 1$. Тогава твърдението е тривиално изпълнено.
Поддръжка. Нека твърдението е изпълнено за някое непоследно достигане на проверката за край на цикъла. Тогава всички диади са в $\texttt{arr}[\texttt{left} \ldots \texttt{right}]$ и $\texttt{left} < \texttt{right}$. Разглеждаме три случая:

1 сл. $\texttt{arr}[\texttt{left}] + \texttt{arr}[\texttt{right}] = \texttt{target}$ – това няма как да е изпълнено понеже достигането е нефинално

2 сл. $\texttt{arr}[\texttt{left}] + \texttt{arr}[\texttt{right}] < \texttt{target}$ – тогава понеже $\texttt{arr}$ е сортиран, за всяко $\texttt{left} + 1 \leq \texttt{i} \leq \texttt{right}$ имаме, че $\texttt{arr}[\texttt{left}] + \texttt{arr}[\texttt{i}] \leq \texttt{arr}[\texttt{left}] + \texttt{arr}[\texttt{right}] < \texttt{target}$. Това означава, че $\texttt{left}$ не участва в никоя диада във $\texttt{arr}[\texttt{left} \ldots \texttt{right}]$. Така получаваме, че всички диади се намират в $\texttt{arr}[\underbrace{\texttt{left} + 1}_{\texttt{left}_{\texttt{new}}} \ldots \texttt{right}]$

3 сл. $\texttt{arr}[\texttt{left}] + \texttt{arr}[\texttt{right}] > \texttt{target}$ – напълно дуален на 2 сл.

Терминация. От цикъла винаги ще излезем, защото или ще открием `val`, или `right` − `left` ще намалява, докато не стане `0`. Излизането става по два начина:

- не е изпълнено условието на ред 5 т.е. `left` ≥ `right` – тогава няма диади в `arr`, и алгоритъмът коректно ще върне `false`.

- изпълнено е условието на ред 9 т.е. `arr`[`left`] + `arr`[`right`] = `target` – тогава алгоритъмът връща `true` т.е. точно това, което искаме.

Двата алгоритъма са доста подобни, използват една често срещана техника за търсене в дадено множество от елементи. Търсенето започва с цялото множество и то постепенно се смалява. Разбира се, тук разгледаните алгоритми имат разлика в сложността, поради разликата в стесняването:

- първият алгоритъм има сложност $O(\log(n))$ понеже винаги разликата между `left` и `right` намалява двойно

- вторият алгоритъм има сложност $O(n)$ понеже винаги разликата между `left` и `right` намалява с единица

Задачи

Задача 1. Да се напише колкото се може по-бърз алгоритъм, който приема масив от различни цели числа $A[1 \dots n]$ с $n \geq 3$, за който има $1 < i < n$ такова, че $A[1 \dots i]$ е сортиран възходящо и $A[i \dots n]$ е сортиран низходящо, и връща това i. След това да се докаже неговата коректност, и да се изследва сложността му по време.

Задача 2. Да се напише колкото се може по-бърз алгоритъм, който при подаден сортиран целочислен масив $A[1 \dots n]$ и число t, което се намира в масива, връща най-малкият и най-големият индекс, на които t се намира в $A[1 \dots n]$. След това да се докаже неговата коректност, и да се изследва сложността му по време.

Задача 3. Да се напише колкото се може по-бърз алгоритъм, който при подаден сортиран целочислен масив $A[1 \dots n]$ и числа $k \geq 2$ и t, връща дали има $0 \leq i_1 < i_2 < \dots < i_k \leq n-1$, за които:

$$A[i_1] + A[i_2] + \dots + A[i_k] = t$$

След това да се докаже неговата коректност, и да се изследва сложността му по време.

Задача 4. За $a, b, x \in \mathbb{Z}$ казваме, че a е по-близо от b до x, ако:

$$|a - x| < |b - x| \lor (|a - x| = |b - x| \;\&\; a < b)$$

Да се напише колкото се може по-бърз алгоритъм, който приема целочислен масив $A[1 \dots n]$, и връща най-близките k на брой числа до t в $A[1 \dots n]$. След това да се докаже неговата коректност, и да се изследва сложността му по време. Може ли да се напише по-бърз алгоритъм при предположение че $A[1 \dots n]$ е сортиран?

Задача 5. Да се напише колкото се може по-бърз алгоритъм, който приема масив $A[1 \dots n]$, съставен от числата $0, 1$ и 2, и го сортира. След това да се докаже неговата коректност, и да се изследва сложността му по време.

Задача 6. Да се напише колкото се може по-бърз алгоритъм, който приема масив $I[1 \dots n]$, съставен от двойки числа, които представят някакъв затворен интервал от цели числа ($[a, b]$ за някои $a, b \in \mathbb{Z}$), и връща нов масив, в който са сляти всички интервали с непразно сечение. След това да се докаже неговата коректност, и да се изследва сложността му по време.

Задача 7. Да се напише колкото се може по-бърз алгоритъм, който приема целочислен масив $A[1 \dots n]$, и връща нов масив от квадратите на $A[1 \dots n]$, който е сортиран. След това да се докаже неговата коректност, и да се изследва сложността му по време.

Задача 8. Да се напише колкото се може по-бърз алгоритъм, който приема целочислен масив $A[1 \dots 2n]$, и връща:

$$\min\{\max\{A'[2i] + A'[2i-1] \mid 1 \leq i \leq n\} \mid A'[1 \dots n] \text{ е пермутация на } A[1 \dots n]\}$$

След това да се докаже неговата коректност, и да се изследва сложността му по време.

Задача 9. Да се напише колкото се може по-бърз алгоритъм, който приема два целочислени масива $A[1 \dots n]$ и $B[1 \dots n]$, и връща:

$$\min\{\sum_{i=1}^{n} |A'[i] - B'[i]| \mid A'[1 \dots n] \text{ е пермутация на } A[1 \dots n] \text{ и } B'[1 \dots n] \text{ е пермутация на } B[1 \dots n]\}$$

След това да се докаже неговата коректност, и да се изследва сложността му по време.